به نام خدا

عنوان :
آموزش ۸۰۵۱ به زبان اسمبلی

نویسنده :
علی حیاتی

کلمات کلیدی : اسمبلی – ۸۰۵۱ – میکرو کنترلر

مقدمه:

در این مقاله قصد داریم به معرفی خانواده ی ۸۰۵۱ به زبان اسمبلی را آموزش دهیم. همانطور که می دانید در بسیاری از دانشگاه ها درس میکروکنترلر ۸۰۵۱ مورد بررسی قرار میگرد و دلیل آن این است که این خانواده را پایه و اساس میکروپرسسور ها و ریزپردازنده ها می دانند به دلیل سادگی این آی سی هنوز در بسیاری از دانشگاه ها تدریس می شود.

پایه و اساس کار میکروکنترلر ها بر اساس زبان ماشین (اسمبلی) است. شما اگر زبان اسمبلی را به طور کامل یاد بگیرید تقریبا زبان ها دیگر را به راحتی یاد می گیرید. امروزه اکثر افراد از زبان های سطح بالا (c و بیسیک و ...) استفاده میکنند و کسی به دنبال یادگیری زبان اسمبلی نمی رود.

قابلیت های میکرو پروسسور:

۱- دستورات متنوع و همه منظوره
۲- قابلیت گسترش بالا
۳- نیازمند تراشه های جانبی برای Interface با دیتای اطراف
۴- دستورات در چند سیکل اجرا می شود.
۵- دستورات نیاز به حافظه ی زیادی دارد.

قابلیت های میکروکنترلر:

۱- دستورات کنترلی زیاد
۲- دستورات پردازشی کم
۳- قابلیت گسترش متوسط رو به بالا دارد
۴- دارای Interface های کنترلی
۵- دستورات در یک یا چند سیکل اجرا می شوند.
۶- دستورات حافظه ای reg و بیتی

قابلیت های DSB :

در کارهای پردازش سیگنال استفاده می شود. مثل فیلتر کردن – تبدیل فوریه گرفتن که نیازمند دستورات پردازشی (ضرب و جمع و ...) است.

۱- نیازمند دستورات پردازشی توانمند
۲- قابلیت گسترش متوسط رو به پایین
۳- نیازمند Interface می باشد و به تراشه های جانبی احتیاج دارد.
۴- دستورات عموما در یک سیکل اجرا می شوند
۵- دستورات رجیستری دارند
۶- چون دستورات پردازشی پیچیده دارند نیازمند کامپایلر هستند.

بررسی زبان اسمبلی با زبان های سطح بالا:

امروزه بسیار از افراد با میکروکنترلر خانواده ی avr,pic کار می کنند و کسی دنبال کار کردن با 8051 نمی رود. چون زبان میکروکنترلر avr و pic سطح بالا است و توابع آماده ای برای استفاده کردن کاربر وجود دارد دیگر کسی سراغ ۸۰۵۱ به زبان اسمبلی نمی رود.
الان اگر من از شما بپرسم چند نفر می توانند بگویند که در lcd چگونه یک نوشته نمایش داده می شود کسی نمی داند زیرا از توابع آماده استفاده می کند و هیچ وقت کار lcd را درک نمی کند ولی در زبان اسمبلی شما باید ابتدا lcd را کامل بشناسی و با ساختار lcd کاملا آشنا شوید بعد شروع به نوشتن برنامه کنید.
من قصد دارم در این مقاله زبان اسمبلی و خانواده ی ۸۰۵۱ را با مثال های کاربردی توضیح دهم.

معیارهای انتخاب میکروکنترلر:

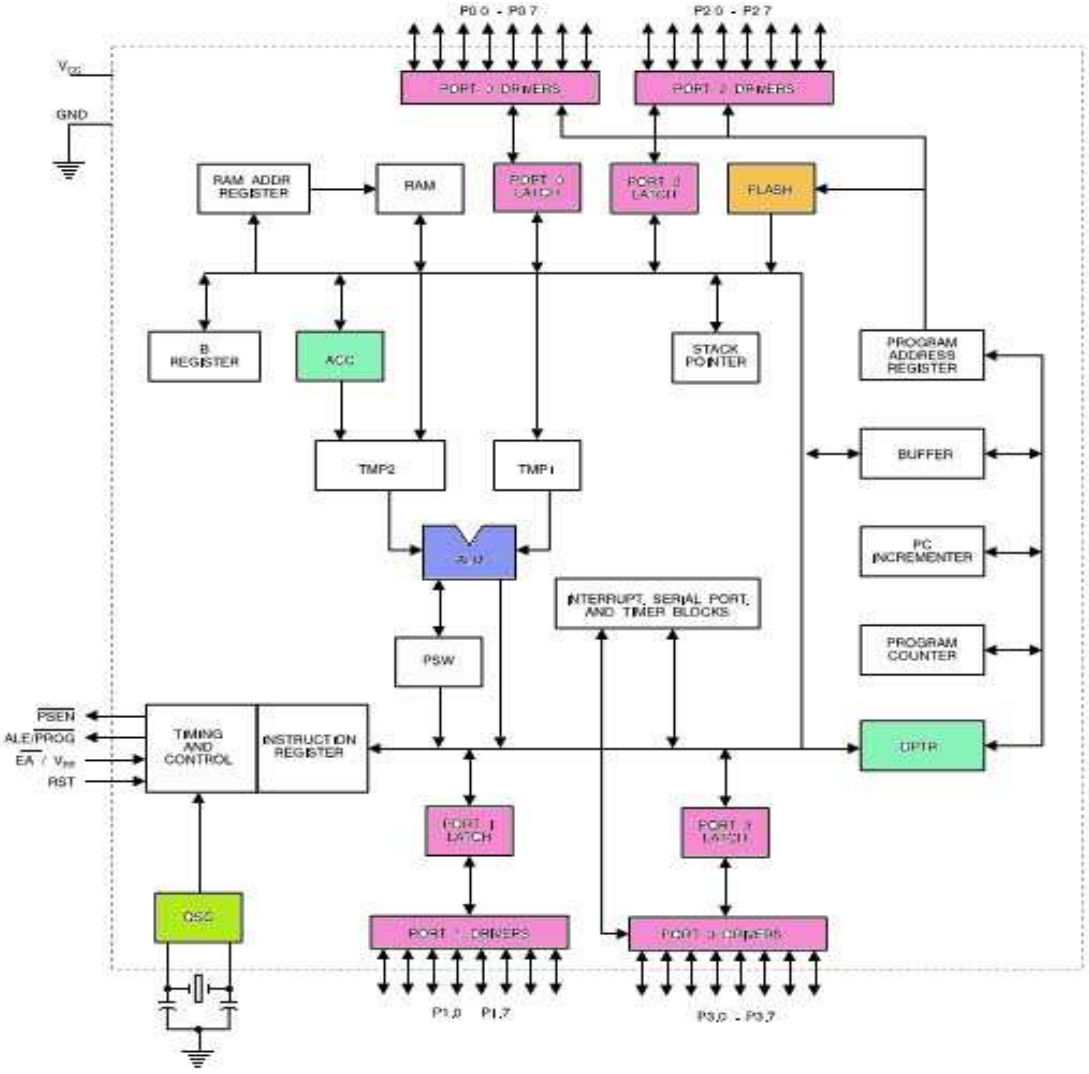
- مهمترین معیار در انتخاب میکروکنترلر این است که بتواند خواسته شما را برآورده کند و آنچه که شما از میکروکنترلر می خواهید را انجام دهد.
- از لحاظ قیمت مناسب باشد شما باید میکرو کنترلری که انتخاب می کنید از لحاظ حافظه و I/O و ... مناسب کار شما باشد.
- مصرف انرژی و جریانی که از منبع می کشد و توانی که مصرف میکند را باید در نظر بگیرید چون در یک جایی شما دستگاهی می سازید که با باطری کار می کند پس باید این پارامترها را درنظر بگیرید.
- مقدار Ram و Rom را باید در نظر داشت.

خانواده ی ۸۰۵۱ :

تاریخچه:

این میکروکنترلر در سال ۱۹۸۱ توسط شرکت اینتل ساخته شده است این میکروکنترلر ۸ بیتی است. این میکروکنترلر دارای 128 بایت Ram و 4k بایت Rom و دوتایمر و یک درگاه سریال و چهار پورت (هر پورت دارای ۸ بیت) است. دلیل این که این خانواده را ۸ بیتی می نامند این است که Cpu آن می تواند فقط بر روی ۸ بیت داده کار کند. اگر داده بزرگتر از ۸ بیت بود باید به داده های ۸ بیتی تقسیم شود مثلا یک داده ی ۱۶ بیتی باید به دو تا ۸ بیتی تقسیم شود.

میکرو کنترلر At89C51 عضو اصلی خانواده ی ۸۰۵۱ است.



ساختمان داخلی At89CXX (شکل ۱)

Port :

پورت ها یا I/o به عنوان ورودی و خروجی استفاده می شود. هر پورت دارای ۸ بیت است که ما می توانیم به صورت پورت از آن استفاده کنیم یا به صورت بیت.
همانطور که در شکل یک می بینید این آی سی دارای ۴ پورت است که برای هر پورت یک اسم اختصاص داده شده است (P0,P1,P2,P3)
هر پورت را می توان به صورت بیتی هم استفاده کرد (مثلا P1.0,P1.1 و...)
کاربر برای کنترل دستگاه ها و ارسال اطلاعات باید از پورت ها استفاده کند.

ALU :

ALU یعنی واحد محاسبه و منطقی.
تمام عملیات محاسباتی (جمع و تفریق و ضرب و...) و منطقی (AND و OR و...) در این قسمت انجام می پذیرد.

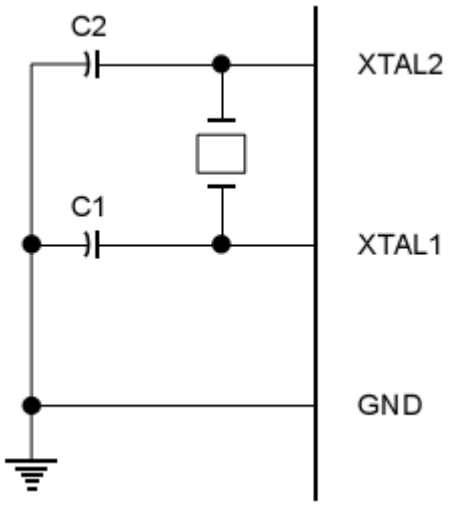
FLASH :

برنامه ای که کاربر می نویسد و بعد از کامپایل کردن فایل هگز آن را پروگرام می کند برنامه در این قسمت از حافظه ریخته می شود.
شما وقتی میکروکنترلر را RESET می کنید از آدرس H۰۰ حافظه فلش برنامه شروع می شود یعنی با قطع شدن برق شهر این حافظه پاک نمی شود و برنامه دوباره اجرا می شود.

OSC :

در داخل میکروکنترلر ها یک بلوکی (شکل۱) به نام اسیلاتور وجود دارد که یک کلاک برای میکروکنترلر تهیه می کند تا میکروکنترلر بتواند کار پردازش را انجام دهد.
فرکانس این اسیلاتور توسط کاربر قابل تنظیم است. در میکروکنترلر ها دو پایه ی سخت افزاری وجود دارد به نام Xtal1 و Xtal2 که فرد می تواند کریستال مورد نظر را که بر حسب MHZ است وصل کند.
جهت محاسبه ی کلاک داخلی میکروکنترلر می توان از فرمول زیر استفاده کرد.

$$F = 1/12 * \text{کریستال}$$
$$T = 1/F$$



خازن های C1 و C2 جهت پایداری اسیلاتور مورد استفاده قرار می گیرند و می توانند بین 30pf یا 33pf باشند.

ACC:

این رجیستر ۸ بیتی است و همانطور که در شکل یک می بینید نتیجه ی تمام عملیات محاسباتی و منطقی در این رجیستر است و تنها این رجیستر با قسمت ALU در ارتباط است.

DPTR:

این رجیستر ۱۶ بیتی است.

تنها رجیستر ۱۶ بیتی در خانواده 8051 رجیستر Dptr است . یکی از کاربرد های مهم قرار دادن جدول(Data Base) در داخل این رجیستر است.

PSW:

بیت های پرچم است جهت انتخاب بانک هاو...

بررسی سری معروف آی سی های خانواده ۸۰۵۱:

نام آی سی	ROM	RAM	I/O	تایمر	وقفه	VCC
AT89C51	4K	128	32	2	6	5
AT89C52	8K	128	32	3	8	5
AT89C55	20K	128	32	3	8	5
AT89C2051	2K	128	15	2	6	3

حافظه Ram :

شما باید آدرس های حافظه ی Ram بشناسید و بدانید کدام حافظه ها قابل دسترسی است.

به شکل زیر توجه کنید:

آدرس	خانه هاي 8 بتي RAM								ثبات	عملكرد
FF										
F0	F7	F6	F5	F4	F3	F2	F1	F0	B	ثبات B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC	ثبات A يا انباره
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW	كلمه وضعيت
B8	--	--	--	BC	BB	BA	B9	B8	IP	
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3	پورت 3
A8	AF	--	--	AC	AB	AA	A9	A8	IE	كنترل وقفه ها
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2	پورت 2
99	قابل آدرس دهي نيست								SBUF	ارتباط سريال
98	9F	9E	9D	9C	9B	9A	99	98	SCON	
90	97	96	95	94	93	92	91	90	P1	پورت 1
8D	قابل آدرس دهي نيست								TH1	بايت سنگين تايمر 1
8C	قابل آدرس دهي نيست								TH0	بايت سنگين تايمر 0
8B	قابل آدرس دهي نيست								TL1	بايت سبك تايمر 1
8A	قابل آدرس دهي نيست								TL0	بايت سبك تايمر 0
89	قابل آدرس دهي نيست								TMOD	مد تايمر
88	8F	8E	8D	8C	8B	8A	89	88	TCON	مد شمارنده
87	قابل آدرس دهي نيست								PCON	
83	قابل آدرس دهي نيست								DPH	بايت سنگين ثبات DPTR
82	قابل آدرس دهي نيست								DPL	بايت سبك ثبات DPTR
81	قابل آدرس دهي نيست								SP	اشاره گر پشته
80	87	86	85	84	83	82	81	80	P0	پورت 0
7Fتا30	80 بايت براي خواندن و نوشتن موقت									فقط بايتي
2Fتا20	16 بايت براي خواندن و نوشتن موقت									بيتي و بايتي
1Fتا18	بانك 3								R0-R7	بانك هاي ثباتي شامل R0-R7
17تا10	بانك 2								R0-R7	
0Fتا08	بانك 1								R0-R7	
07تا00	بانك 0								R0-R7	
MGH	MGH								MGH	MGH

جدول تقسيم بندی حافظه Ram

۸۰۵۱ دارای ۱۲۸ بایت حافظه Ram است.

۱- آدرس 00h تا 1Fh که ۳۲ بایت است برای بانک های ثبات و پشته کنار گذاشته شده است.

۲- از 20h تا 2fh برای آدرس دهی بیتی در برنامه مورد استفاده قرار می گیرد.

۳- از 30h تا 7fh جهت خواندن و نوشتن بایتی در برنامه مورد استفاده قرار می گیرند.

نکته:

در خانه هایی که نوشته شده است قابل آدرس دهی نیست یعنی قابل خواند و نوشتن توسط کاربر نیست.

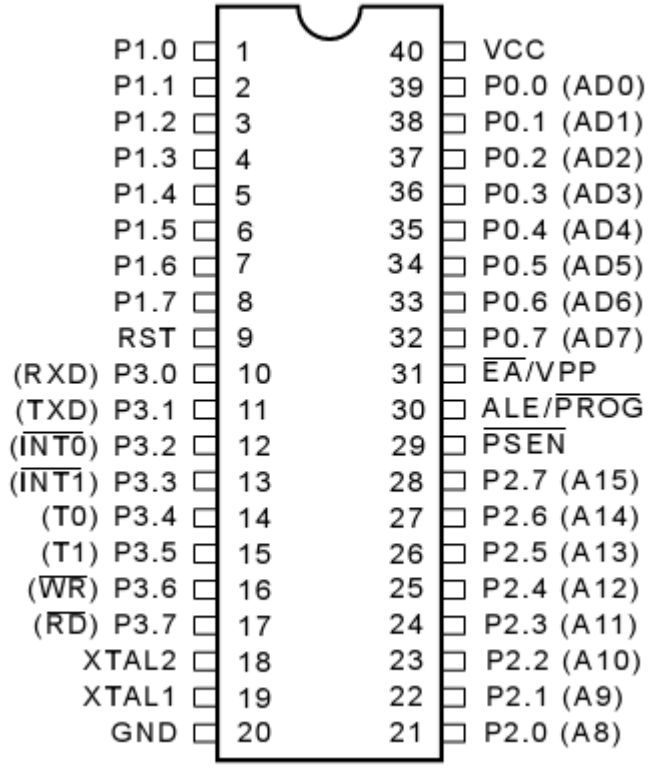
حافظه ها

Rom (Read only Memory): حافظه ی فقط خواندنی است و نمی توان چیزی داخل آن نوشت. با قطع برق اطلاعات Rom پاک نمی شود.

E2prom : حافظه فقط خواندنی است که قابلیت برنامه ریزی دارد و می تواند اطلاعات را در خود نگاه دارد این اطلاعات با قطع برق از بین نمی رود.

Ram (Read Accesory Memory): حافظه خواندنی و نوشتنی است و با قطع برق اطلاعات از بین می رود.داده ایی که همواره مورد نیاز است و نیاز داریم که همواره از آن استفاده کنیم در داخل Ram می گذاریم.

پایه های ۸۰۵۱:



پایه های آی سی AT89C51 & AT89C52

همانطور که در شکل بالا می بینید این آی سی ۴۰ پایه است که ۳۲ پایه ی آن به عنوان I/O استفاده شده و دارای دوپایه جهت تغذیه آی سی است که تغذیه ی این آی سی ۵ ولت است و دو پایه جهت اتصال کریستال و ۳ پایه کنترلی وجود دارد.

پورت ها:

این آی سی شامل ۴ پورت به نام های P0 و P1 وP2 وP3 است هر پورت شامل ۸ بیت است که کاربر می تواند به صورت پورت یا به صورت بیت از این پایه ها استفاده کند.
مثلا P1 شامل ۸ بیت است که P1.0 به عنوان کم ارزش ترین بیت(LSB) و P1.7 به عنوان پر ارزش ترین بیت (MSB) مورد استفاده قرار می گیرند.

Rst :

این پایه جهت Reset کردن میکروکنترلر در مواقعی که میکرو هنگ می کند یا جهت Refresh کردن میکرو مورد استفاده قرار می گیرد.
این پایه فعال یک است یعنی اگر شما این پایه را به صورت لحظه ای به VCC متصل کنید میکرو Reset می شود. پس در حالت عادی این پایه زمین است.

ALE :

اگر از RAM یا ROM بیرونی استفاده کنیم این پایه پورت صفر را به عنوان خطوط آدرس معرفی می کند.

PSE :

این پایه برای زمانی است که بخواهیم از RAM یا ROM بیرونی استفاده کنیم که بسته به شرایط یا صف می شود یا یک.

EN :

اگر این پایه را یک کنیم میکرو برنامه را از ROM داخلی خودش شروع به خواندن می کند و اگر این پایه را صفر کنیم میکرو از ROM داخلی خودش هیچ اطلاعاتی نمی خواند و با برنامه ریزی که شده از ROM بیرونی شروع به خواندن اطلاعات می کند.

RXT (P3.0): پایه دریافت اطلاعات سریال
TXT (P3.1): پایه ارسال داده های سریال
INT0 (P3.2): وقفه خارجی صفر
INT1 (P3.3): وقفه خارجی یک

وظایف پردازنده :

۱- عملیات محاسباتی و منطقی به عبارتی دستکاری داده ها
۲- نقل و انتقال داده (ذخیره سازی یعنی اطلاعات را از پورت داخل حافظه قرار داد یا از حافظه داخل پورت قرار دهد یا از رجیستر ها منتقل کند).
۳- کنترل روال برنامه (وقتی از الگوریتم استفاده می شود دستورات شرطی مطرح می شود).
۴- کنترل رفتار ماشین (وقفه یک فرآیند سخت افزاری است و اینکه آیا پردازنده به وقفه جواب بدهد یا نه جزء کنترل رفتار ماشین است).

آموزش برنامه نویسی اسمبلی برای ۸۰۵۱ :

رجیستر های ۸۰۵۱ :

در ۸۰۵۱ در ۴بانک ثبات وجود دارد که داخل هر بانک ثبات های ۸ بیتی R0 تا R7 وجود دارد که در حالت پیش فرض بانک صفر فعال است. شما با مقدار دهی Psw.3 و Psw.4 بانک های مورد نظر را که در جدول زیر مشخص شده است می توانید انتخاب کنید.

	RS1(PSW.4)	RS0(PSW.3)
بانك صفر	0	0
بانك يك	0	1
بانك دو	1	0
بانك سه	1	1

شما با صفر و یک کردن این دو بیت می توانید بانک مورد نظر را انتخاب کنید.
Psw یک بایت کنترلی است که شامل ۸ بیت است که بیت سوم و چهارم جهت انتخاب بانک مورد استفاده قرار می گیرد.

دستور Mov :

Mov به معنای حرکت دادن است و در Ram داخلی عمل می کند. طریقه ی استفاده آن به صورت زیر است:

Mov مقصد , مبدا

در این دستور مقدار مبدا در داخل مقصد کپی می شود ولی مقدار مبدا پاک نمی شود. این دستور را می توان به روش های زیر مورد استفاده قرار داد:

```
Mov  R1,#255
```

در دستور بالا مقدار ۱۰ در داخل ثبات R1 قرار می گیرد. وقتی یک عدد را به صورت مستقیم می خواهیم در یک رجیستر بریزیم باید از علامت # استفاده کنیم.

```
Mov  R1,#0FFh
```

در این دستور مقدار مقصد یک عدد در مبنای هگزا است چون از نماد H که بیانگر عدد در مبنای هگزا است استفاده شده است. دراین دستور مقدار FFh که معادل عدد 255 است را در داخل R1 می ریزد.

آن صفر قبل از F برای آن است که کامپایلر آن را حرف نگیرد.

```
Mov  R1,#11111111b
```

این دستور مقدار مقصد را که یک عدد باینری است را در ثبات R1 قرار می دهد. معادل این عدد در مبنای دهدهی برابر عدد ۲۵۵ است. همانطور که در مثال های بالا می بینید فرقی نمی کند که در چه مبنایی بنویسید.

```
Mov  R1,#100
Mov  R2,R1
```

ابتدا مقدار ۱۰۰ در داخل R1 قرار داده می شود و در دستور بعد مقدار R1 در داخل R2 ریخته می شود به این روش روش غیر مستقیم گفته می شود.

هنگامی که می خواهیم اطلاعات یک ثبات را در داخل ثبات دیگر بریزیم نیازی به استفاده از علامت # نیست.

```
Mov  C,BIT
Mov  Bit,C
```

این دستور مقدار Bit را در داخل کری قرار می دهد و بالعکس.

دستور Move :

```
Move   A,100
```

این دستور محتوای خانه ۱۰۰ حافظه را در داخل آکامولاتور قرار می دهد.(روش آدرس دهی مستقیم)

```
Move  R1,#100
Move  A,@R1
```

ابتدا مقدار ۱۰۰ را در داخل R1 قرار می دهد و بعد محتوای آدرس R1 را در داخل آکامولاتور قرار می دهد.(آدرس دهی غیر مستقیم)

دستور Movc :

این دستور در Rom عمل می کند یعنی یک کد ثابت را دارد و نتیجه ی این دستور در آکامولاتور قرار می گیرد.

```
Movc  A,@A+DPTR
Movc  A,@A+PC
```

دستور Movx :

این دستور در Ram خارجی عمل می کند. نتیجه ی این دستور هم در آکامولاتور است.

```
Movx   A,@Ri
```

آدرسی که Ri اشاره می کند از Ram خارجی در داخل آکومولاتور قرار بده.

```
Movx   A,@DPTR
```

این دستور برای آدرس دهی ۱۶ بیتی استفاده می شود.

دستور Setb :

```
Setb   bit
```

این دستور برای یک کردن(یک منطقی) یک بیت مورد استفاده قرار میگیرد.

مثال:

```
Setb       p2.0   ;این دستور پورت دو بیت صفر را یک میکند(۵+ ولت)
```

دستور Clr :

```
Clr      bit
```

این دستور برای صفر کردن (صفر منطقی) یک بیت مورد استفاده قرار می گیرد.

مثال:

```
Clr       p2.0   ;این دستور پورت دو بیت صفر را صفر میکند(صفر ولت)
```

دستور CPL :

```
CPl    bit
```

این دستور از بیت متمم می گیرد یا بیت را معکوس می کند.

مثال:

```
Setb  p2.0
CPl   p2.0    ;این دستور از پورت دو بیت صفر متمم می گیرد یعنی این بیت را صفر می کند
```

دستور Jmp :

Jmp Lable

وقتی برنامه به این دستور برسد به برچسب که تعریف شده است پرش می کند.برچسب هر اسمی می تواند باشد ولی اولین کلمه نباید عدد باشد چون کامپایلر خطا می گیرد.

مثال:

```
Main:
Clr  p2.0
Setb  p2.0
Jmp  main  ; پرش به برچسب
```

همانطور که در برنامه بالا می بینید وقتی برنامه به این دستورjmp برسد به برچسبmain پرش می کند و برنامه را دوباره تکرار می کند.

دستور Inc :

Inc Var

این دستور باعث افزایش Val به اندازه ی یک واحد می شود.

مثال:

```
Mov  R1,#10
Inc  R1
```

این برنامه مقدار R1 را برابر ۱۰ قرار می دهد و بعد مقدار R1 را به اندازه یک واحد افزایش می دهد بعنی بعد از این دستور مقدار R1 برابر با ۱۱خواهد شد .

دستور Dec :

Dec Var

این دستور باعث کاهش Val به اندازه یک واحد می شود.

مثال:

```
Mov  R1,#10  ; مقدار ۱۰ را داخل ثبات می ریزد
Dec  R1      ; یک واحد از ثبات کم میکند
```

دستورات منطقی

دستور ANL (AND):

AND داده ۸بیتی , داده ۸ بیتی

این دستور دوتا داده ۸بیتی را با هم And می کند.این دستور بیت به بیت با هم And می شوند.

مثال:

```
Mov  a,#56h
ANL  a,25h     ; A=4H
```

56H	0	1	0	1	0	1	1	0
25H	0	0	1	0	0	1	0	1
OUT=4H	0	0	0	0	0	1	0	0

A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

همانطور که در جدول بالا می بینید بیت ها تک تک با هم And می شوند اگر هر دوبیت یک باشند خروجی یک می شود در غیر این صورت خروجی صفر است.

دستور ORL (OR):

ORL داده ۸ بیتی , داده ۸ بیتی

این دستور دو تا داده ۸بیتی را با هم OR می کند.

مثال:

```
Mov  a,#56h
ORL  a,25h
```

56H	0	1	0	1	0	1	1	0
25H	0	0	1	0	0	1	0	1
OUT=77H	0	1	1	1	0	1	1	1

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

همانطور که در جدول بالا می بینید بیت ها تک تک با هم OR می شوند اگر یکی از دو بیت یک باشند خروجی یک خواهد شد.

دستور XRL :

XRL داده ۸ بیتی , داده ۸ بیتی

این دستور دوتا داده ۸ بیتی را با هم XOR می کند.

مثال:

```
Mov  a,#56h
XRL  a,25h
```

56H	0	1	0	1	0	1	1	0
25H	0	0	1	0	0	1	0	1
OUT=73H	0	1	1	1	0	0	1	1

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

همانطور که در جدول بالا می بینید بیت ها تک تک با هم XOR می شوند اگر یکی از بیت ها یک باشد و یکی از بیت ها صفر باشد خروجی یک خواهد بود در غیر این صورت خروجی صفر می شود.

دستورات ریاضی

دستور ADD :

این دستور برای جمع دو داده ۸ بیتی مورد استفاده قرار می گیرد. برای عمل جمع حتما باید یکی از داده ها انباره(A) باشد و داده ی دیگر می تواند ثبات یا داده ی فوری باشد. نتیجه ی جمع در انباره(A) قرار می گیرد.

مثال:

```
Mov   A,#20
ADD   A,#10
```

این برنامه عدد ۲۰ را با عدد ۱۰ جمع می کند و نتیجه که عدد ۳۰ است در داخل انباره قرار می گیرد.

دستور ADDC :

این دستور برای جمع دو عدد ۱۶ بیتی مورد استفاده قرار می گیرد البته این کار را مستقیم انجام نمی دهد ابتدا آن دوعدد را به داده های ۸بیتی تجزیه کرد.

ابتدا ۸ بیت اول عدد اول را با ۸ بیت اول عدد دوم جمع می کنیم واگر کری ایجاد شد این کری را در مجموع ۸ بیت دوم هر دو عدد جمع می کنیم.

مثال:

می خواهیم عدد 3CE7+3B8D را در مبنای هگز با هم جمع کنیم. برای جمع ابتدا پرچم کری را صفر می کنیم چون ممکن است از قبل توسط برنامه ای دیگر یک شده باشد. سپس این داده ۱۶ بیتی را به داده ۸ بیتی تجزیه می کنیم. می دانیم که در میکر همه چیز ۸بتی می باشد پس از جمع دو داده ۱۶ بیتی مطمئنا یک داده ۱۶ بزرگتر تولید می شود برای همین ما بایت سبک را در R1قرار می دهیم و بایت سنگین را در R2 قرار می دهیم البته این یک فرض دلخواه می باشد شما می توانید این داده را در هر ثباتی قرار دهید و از حاصل این جمع در جایی دیگر استفاده کنید.

```
CLR   C
MOV   A,#0E7H
ADD   A,#8DH
MOV   R1,A
MOV   A,#3CH
```

```
ADDC   A,#3BH
MOV    R2,A
```

دستور SUBB :

این دستور برای تفریق کردن دو عدد ۸ بیتی مورد استفاده قرار می گیرد. برای عمل تفریق باید یکی از داده ها باید انباره(A) باشد و عدد دوم می تواند ثبات یا داده ی فوری باشد.

مثال:

```
Mov    R1,#20
Mov    A,#30
Subb   A,R1
```

این برنامه عدد ۳۰ را از عدد ۲۰ کم می کند و نتیجه که عدد ۱۰ است در انباره قرار می گیرد.

دستور ضرب (MUL):

```
MUL    AB
```

قالب این دستور به صورت بالا است یعنی وقتی برنامه به این دستور برسد A را در B ضرب می کند و نتیجه را در A قرار می دهد.

البته باید توجه داشت که ضرب این دو عدد نباید از ۸ بیت بیشتر شود.

مثال:

```
Mov   A,#20h
Mov   B,#10h
Mul   AB
```

این برنامه عدد 20h را در عدد 10h ضرب می کند و نتیجه را در انباره می ریزد.

دستور تقسیم (DIV):

```
Div    AB
```

قالب این دستور به صورت بالا است و این دستور A را تقسیم بر B می کند و خارج قسمت در داخل ثبات A و باقیمانده در ثبات B قرار می گیرد.

مثال:

```
Mov   A,#35
Mov   B,#10
Div    AB
```

این دستور عدد ۳۵ را بر عدد ۱۰ تقسیم می کند و خارج قسمت برابر عدد ۳ است داخل انبار ذخیره می شود و باقیماده که برابر عدد ۵ است در ثبات B قرار می گیرد.

دستور CALL:

این دستور برای تعریف زیر برنامه مورد استفاده قرار می گیرد و به صورت زیر تعریف می شود.

```
Call    Lable
```

وقتی برنامه به این دستور برسد به برچسب (Lable) پرش می کند و زیر برنامه را اجرا می کند و با دستور Ret به خط بعدی برنامه که از آنجا فراخوانی شده بر می گردد.

با هر بار Call کردن یک بار زیر برنامه اجرا می شود و دوباره به ادامه ی برنامه ی اصلی بر می گردد.

از این دستور معمولا برای نوشتن تاخیر و ... مورد استفاده قرار می گیرد.

مثال:

```
Mov     R1,#10
Call    L1
Mov     R2,#20
Call    L1

L1:
Mov     R4,#120
Ret
```

دستورات شرطی:

دستور JB :

پرش در صورت یک بودن بیت.

JB Bit,Lable

در صورتی که بیت یک باشد به برچسب پرش می کند در غیر این صورت خط بعد از خودش را اجرا می کند.

از این دستور برای چک کردن یک بیت از میکرو مورد استفاده قرار می گیرد.

مثال:

```
UP:
JB  P2.0,L1
JMP  UP
L1:
SETB  P0.0
```

در این برنامه P2.0 را چک می کند در صورت یک بودن به L1 پرش می کند در غیر اینصورت پرش به UP می کند..

دستور JNB :

پرش در صورت صفر بودن.

JNB Bit,Lable

در صورتی که بیت صفر باشد به برچسب پرش می کند در غیر این صورت خط بعد از خودش را اجرا می کند.

از این دستور برای چک کردن یک بیت از میکرو مورد استفاده قرار می گیرد.

مثال:

```
UP:
JNB  P2.0,L1
JMP  UP
L1:
SETB  P0.0
```

در این برنامه P2.0 را چک می کند و در صورت صفر بودن این بیت به L1 پرش می کند در غیر این صورت به UP می رود.

دستور JC (Jump Cary) :

JC Lable

در صورتی که کری یک باشد به Lable پرش می کند.

دستور JNC (Jump Not Cary) :

JNC Lable

در صورتی که کری صفر باشد به lable پرش می کند.

دستور JZ (Jump Zero):

JZ Lable

در صورتی که A برابر صفر باشد به lable پرش می کند.

دستور JNZ (Jump Not Zero):

JNZ Lable

در صورتی که A نامساوی صفر باشد به Lable پرش می کند.

دستورDJNZ (Decr Jump Not Zero):

DJNZ Register,Lable

این دستور یک واحد از Register کم می کند و در صورت صفر نبودن به Lable پرش می کند و در صورت صفر بودن خط بعدی خود را اجرا می کند.

مثال:

```
Mov   R1,#25
Djnz  R1,$
End
```

در این برنامه مقدار R1 را برابر با ۲۵ قرار می دهد سپس از R1 یک واحد کم می کند و با صفر مقایسه می کند در صورت صفر نبودن به خودش پرش می کند اینقدر از R1 کم می کند که به صفر برسد بعد از صفر شدن R1 برنامه به پایان می رسد.

دستور CJNE (مقایسه):

از این دستور برای مقایسه دو عدد استفاده می شود.

CJNE X,Y,Lable

X و Y با هم مقایسه می شوند در صورت نامساوی بودن به Lable پرش می کند و در صورت برابر بودن به خط بعدی برنامه می رود.

```
CJNE  A,Rn,lable
CJNE   Rn,#data,lable
CJNE  @Ri,#data,lable
```

NOP (No Operation):

این دستور هیچ عملکردی ندارد و برای تلف کردن وقت است.

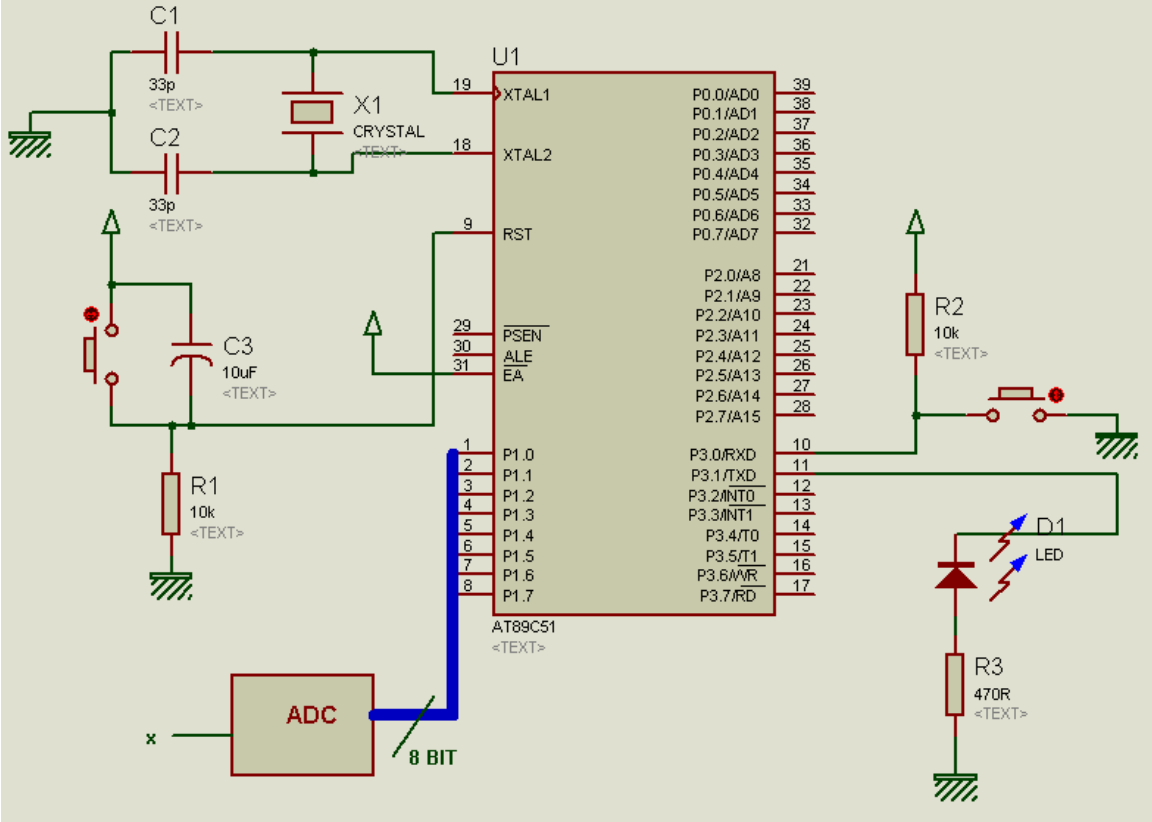
طریقه ی نوشتن تاخیر در برنامه:

در برنامه هایی که توسط ۸۰۵۱ نوشته می شود نیاز است که یک یا چند زیر برنامه برای تاخیر تعریف کرد.
شما می توانید هر تاخیری بر حسب ثانیه یا میلی ثانیه و یا میکرو ثانیه در برنامه تعریف کنید. هر دستوری که شما در ۸۰۵۱ استفاده می کنید دارای ماشین سیکل است که در قالب یک جدول تعریف شده است. این ماشین سیکل بر حسب میکرو ثانیه است و بستگی به کریستالی که شما استفاده می کنید دارد. شما وقتی Clk میکرو را محاسبه کردید به راحتی ماشین سیکل دستورات بدست می آید.
مثلا دستور Mov دو ماشین سیکل است (با فرض کریستال برابر ۱۲ مگاهرتز) کلاک میکرو برابر یک میکرو ثانیه است پس مدت زمانی که طول می کشد دستور Mov در برنامه اجرا شود ۲ میکروثانیه است.
شما با این روش می توانید مدت زمانی را که طول می کشد یک برنامه به صورت کامل اجرا شود را محاسبه کنید.
معمولا در برنامه ها از ماشین سیکل دستورات صرف نظر می کنند.
مثال: برنامه ی یک تاخیر ۵۰۰ میلی ثانیه ای را بنویسید.
500ms = 500000uS = 50*100*100

```
Mov   R1,#50
Up1:
Mov   R2,#100
Up:
Mov   R3,#100
Djnz   R3,$
Djnz   R2,up
Djnz   R1,up1
```

مثال:
می خواهیم سیستمی طراحی کنیم که ۴۰ داده از ورودی نمونه برداری کند و در حافظه ذخیره کند. آغاز نمونه برداری و ذخیره با فشردن یک کلید خواهد بود. اتمام نمونه برداری با روشن شدن یک LED اعلام می شود ؟(برای نمونه گیری از ADC ۸ بیتی استفاده می کنیم)

حل:



همانطور که می بینید در این سخت افزار یک کلید فشاری وجود دارد که به P3.0 متصل شده است این کلید فعال صفر است یعنی پایه میکرو در زمانی که کلید فشار داده نشده یک است (یک منطقی) و در لحظه ای که کلید فشار داده می شود پایه میکرو را صفر می کند. (صفر = روشن و یک= خاموش)

پس از فشرده شدن کلید باید ۴۰ داده ذخیره شوند که داده ها در حافظه ی RAM داخلی ذخیره میشوند.

```
Setb   P3.1
L1:
JB      p3.0,L1
Mov   R1,#40
Mov   R0,#30H
Next:
Mov    A,P1
Mov   @R0,A
Inc     R0
Djnz   R1,Next
Clr     P3.1
```